FIG. 1
PRIOR ART

Metroarea (mID, mName)
Hotel (hID, hName, m_id)
120

110

mID	mName
101	Northwest

hID	hName	m_id
201	Courtyard	101
202	Doubletree	101

FIG. 2
PRIOR ART

200

```
<hotel>
($h = SELECT hName
FROM Hotel
)
        <metro> ($m = SELECT mName
        FROM Metroarea
        WHERE mID = $h.m_id
        )</metro>
</hotel>
```

FIG. 3
PRIOR ART

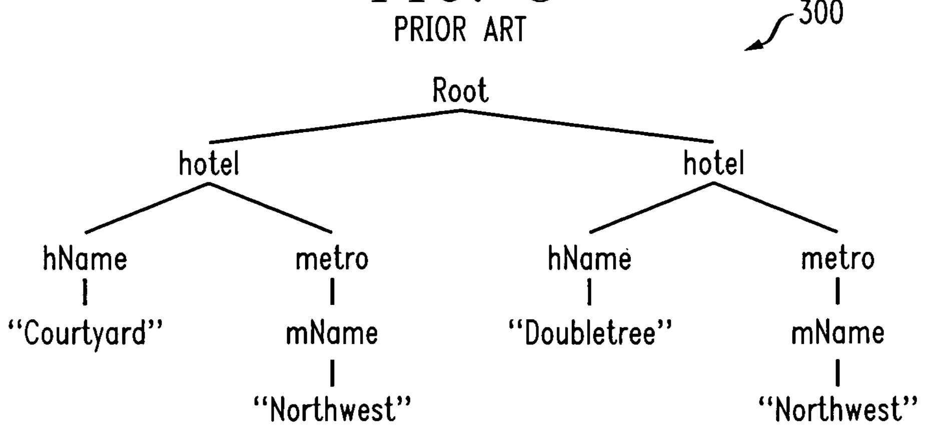


FIG. 4
PRIOR ART

Hotel (hID, hName)
Metroarea (mID, mName, h_id)
420

410

mID	mName	h_id
101	Northwest	201
102	Northwest	202

hID	hName
201	Courtyard
202	Doubletree

FIG. 5

Metroarea (mID, mName)
Hotel (hID, hName, m_id)
Confroom (cID, roomnum,h_id)
500

FIG. 6

600

```
<metro>
($m = SELECT mName
FROM Metroarea)
        <conference-room>
        ($c = SELECT cID, roomnum, m_id
        FROM Confroom, Hotel
        WHERE Confroom.h_id = Hotel.hID
                AND Hotel.m_id = $m.mID
        )</conference-room>
</metro>
```

FIG. 7

Metroarea (mID, mName)
Confroom (cID, roomnum, m_id)

700

FIG. 8

800

Metroarea (mID, mName)
State (sID, sName)
Hotel (hID, hName, starrating, pool, gym, street, city, state_id, metro_id)
Phone (phID, phoneNo)
Confroom (cID, croomnum, capacity, rackrate, c_h_id)
Guestroom (gID, roomnum, type, rackrate, g_h_id)
Availability (aID, startdate, enddate, price, a_r_id)
Restaurant (restID, rName, rCity)

FIG. 9

900

```
<metro>
($m = SELECT mName FROM Metroarea)
   <hotel>
   ($h = SELECT hName, starrating, pool, gym
   FROM Hotel
   WHERE pool > 0 AND metro_id = $m.mID)
      <state>
      ($s = SELECT sName
      FROM State
      WHERE sID = $h.state_id
      )</state>

      <conference-room>
      ($c = SELECT croomnum, capacity
      FROM Confroom
      WHERE rackrate > 2 AND c_h_id = $h.hID)
            <phone-number>
            ($p = SELECT phoneNo
            FROM Phone
            WHERE phID = $h.hID
            )</phone-number>
      </conference-room>

      <guest-room>
      ($g = SELECT roomnum, type
      FROM Guestroom
      WHERE rackrate > 2 AND g_h_id = $h.hID)
            <availability>
            ($a = SELECT startdate, enddate, price
            FROM Availability
            WHERE a_r_id = $g.gID
            )</availability>
      </guest-room>

      <nearby-restaurant>
      ($r = SELECT rName, rCity
      FROM Restaurant
      WHERE rCity = $h.city
      )</nearby-restaurant>
   </hotel>
</metro>
```

FIG. 10

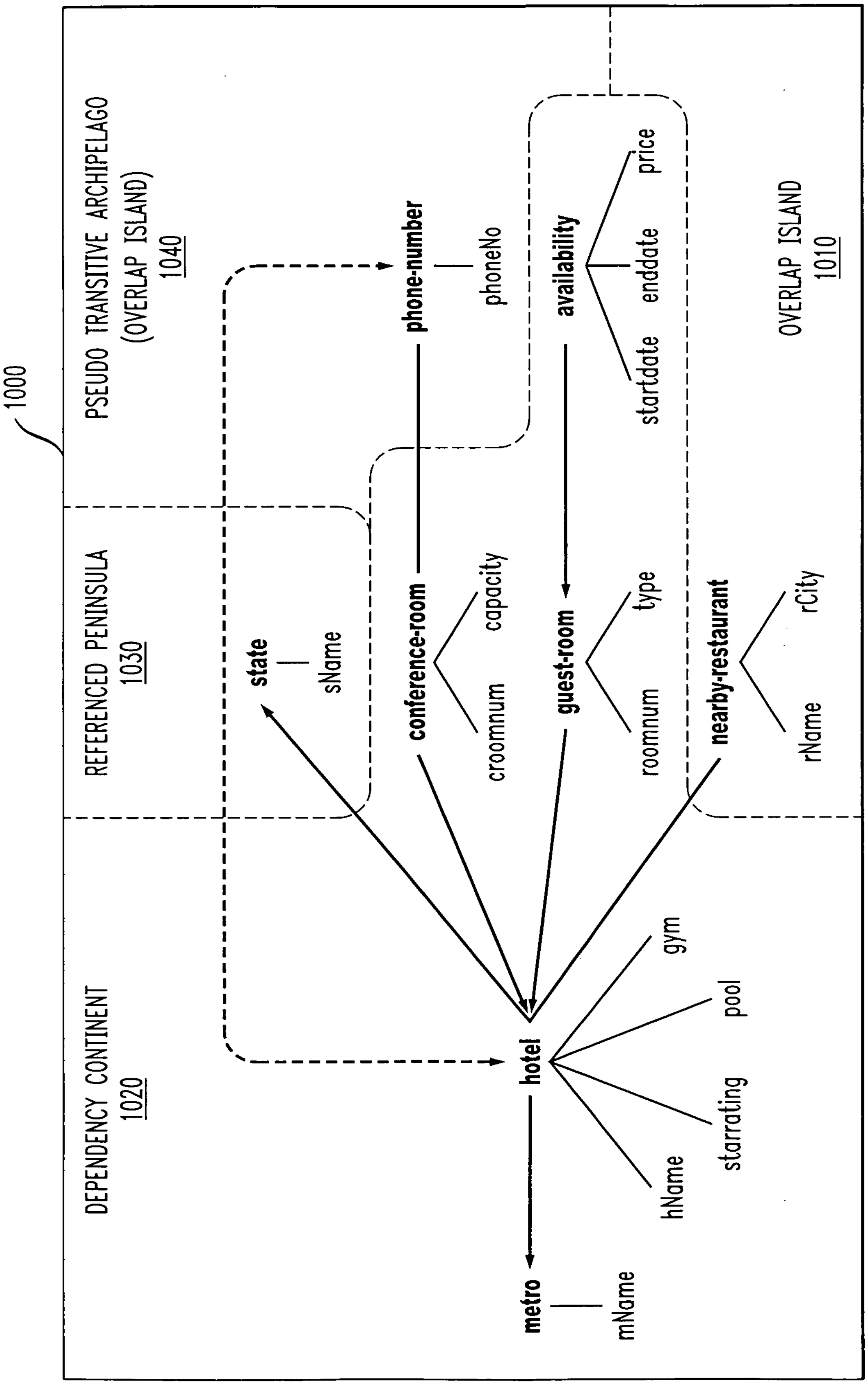

1000
PSEUDO TRANSITIVE ARCHIPELAGO (OVERLAP ISLAND) 1040
REFERENCED PENINSULA 1030
DEPENDENCY CONTINENT 1020
OVERLAP ISLAND 1010
phone-number
phoneNo
availability
startdate
enddate
price
state
sName
conference-room
croomnum
capacity
guest-room
roomnum
type
nearby-restaurant
rName
rCity
hotel
hName
starrating
pool
gym
metro
mName

FIG. 11

NODE CATEGORIZATION PROCESS 1100

```
procedure node-cat-gen(XMLNode node)
begin
1.  if (node shares underlying tables with other nodes && the cardinality
relationship of node and its parent is not 1:n)
2.  then
3.      node is in OI
4.  else
5.      switch (direct parent's category)
6.  case DC:
7.  switch (cardinality relationship of node and its parent)
8.  case 1:1:   node and its child leaf nodes are in DC
9.  case n:1:   node and its child leaf nodes are in DC
10. case 1:n:   node and its child leaf nodes are in RP
11. case m:n:   node and its child leaf nodes are in OI
12. end switch
13. case RP:
14. if (cardinality relationship of node and its parent is m:n)
15. then
16.     node and its child leaf nodes are in OI
17. else
18.     node and its child leaf nodes are in RP
19. case OI:
20. node and its child leaf nodes are in OI
21. end switch
18.for (each child branch node sub of node)
19. node-cat-gen(sub)
end
```

FIG. 12

DELETION TRANSLATION PROCESS

1200

```
procedure node-delete(XMLNode node)
begin
1.  switch (the category of node)
2.  case DC:
3.  if (node is a leaf node) then
4.      if (node is not a required child of its parent) then
5.      for the element base view of its parent, set the corresponding
attribute to NULL
6.  else
7.      node cannot be deleted according to DTD
8.  else
9.      delete the corresponding tuple from element base view
10.     for (each child branch DC-node sub of node)
11. node-delete(sub)
12. case RP:
13. if (node is an RP-root-node) then
14.     if (node is not a required child of its parent) then
15.     for the element base view of its parent, set the corresponding
foreign key to NULL
16. else
17.     node cannot be deleted according to DTD
18. else
19. node cannot be deleted to avoid side-effects
20. case OI:
21. node cannot be deleted to avoid side-effects
22. end switch
end
```

FIG. 13

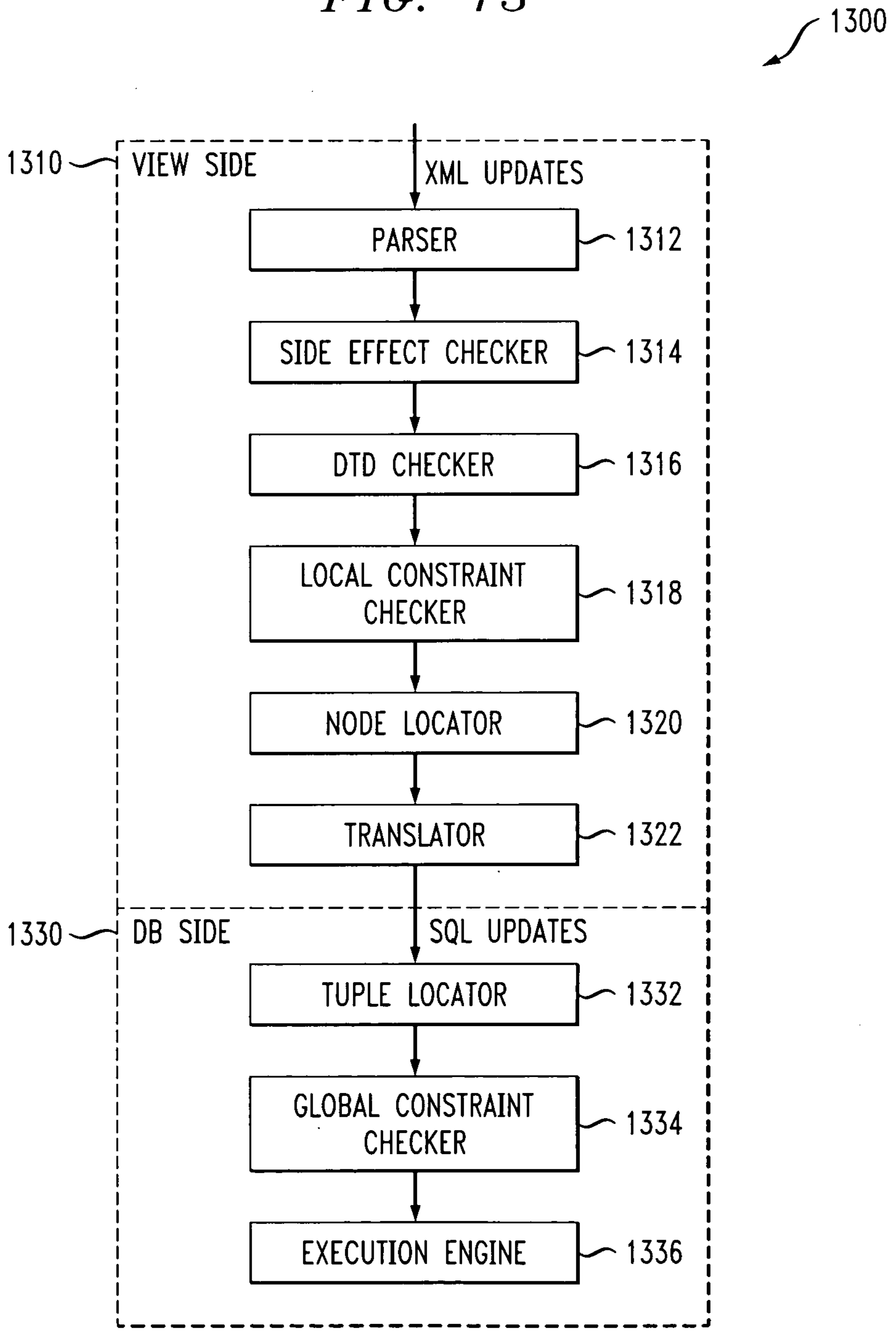

1300
1310
VIEW SIDE
XML UPDATES
PARSER
1312
SIDE EFFECT CHECKER
1314
DTD CHECKER
1316
LOCAL CONSTRAINT CHECKER
1318
NODE LOCATOR
1320
TRANSLATOR
1322
1330
DB SIDE
SQL UPDATES
TUPLE LOCATOR
1332
GLOBAL CONSTRAINT CHECKER
1334
EXECUTION ENGINE
1336

FIG. 14

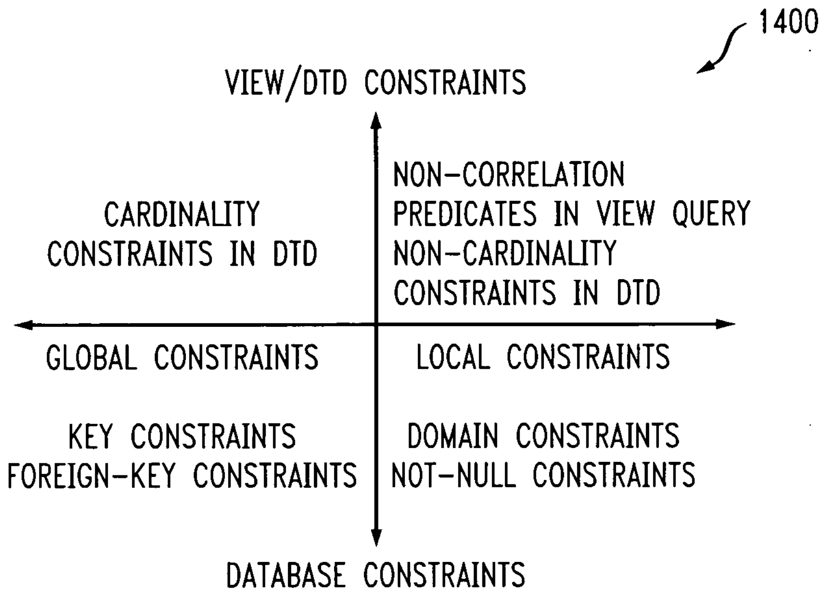
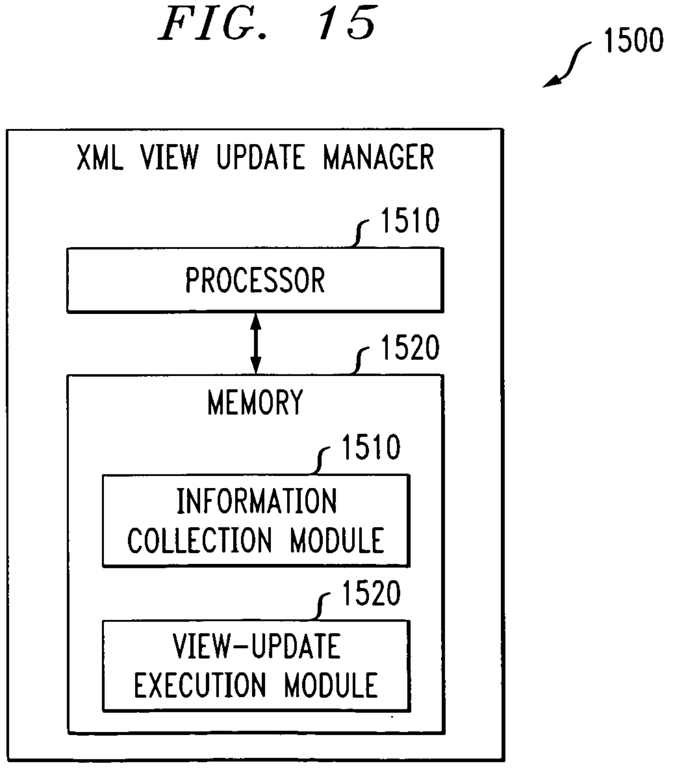

1400
VIEW/DTD CONSTRAINTS
CARDINALITY CONSTRAINTS IN DTD
NON-CORRELATION PREDICATES IN VIEW QUERY
NON-CARDINALITY CONSTRAINTS IN DTD
GLOBAL CONSTRAINTS
LOCAL CONSTRAINTS
KEY CONSTRAINTS
FOREIGN-KEY CONSTRAINTS
DOMAIN CONSTRAINTS
NOT-NULL CONSTRAINTS
DATABASE CONSTRAINTS

FIG. 15


1500
XML VIEW UPDATE MANAGER
1510
PROCESSOR
1520
MEMORY
1510
INFORMATION COLLECTION MODULE
1520
VIEW-UPDATE EXECUTION MODULE